Efficient Probabilistic Diagnostics for Electrical Power Systems

Ole J. Mengshoel*, Mark Chavira†, Keith Cascio‡, Scott Poll§, Adnan Darwiche¶, Serdar Uckun‖

Abstract

We consider in this work the probabilistic approach to model-based dignosis when applied to electrical power systems (EPSs). Our probabilistic approach is formally well-founded, as it based on Bayesian networks and arithmetic circuits. We investigate the diagnostic task known as fault isolation, and pay special attention to meeting two of the main challenges — model development and real-time reasoning — often associated with real-world application of model-based diagnosis technologies. To address the challenge of model development, we develop a systematic approach to representing electrical power systems as Bayesian networks, supported by an easy-to-use specification language. To address the real-time reasoning challenge, we compile Bayesian networks into arithmetic circuits. Arithmetic circuit evaluation supports real-time diagnosis by being predictable and fast. In essence, we introduce a high-level EPS specification language from which Bayesian networks that can diagnose multiple simultaneous failures are auto-generated, and we illustrate the feasibility of using arithmetic circuits, compiled from Bayesian networks, for real-time diagnosis on real-world EPSs of interest to NASA. The experimental system is a real-world EPS, namely the Advanced Diagnostic and Prognostic Testbed (ADAPT) located at the NASA Ames Research Center. In experiments with the ADAPT Bayesian network, which currently contains 503 discrete nodes and 579 edges, we find high diagnostic accuracy in scenarios where one to three faults, both in components and sensors, were inserted. The time taken to compute the most probable explanation using arithmetic circuits has a small mean of 0.2625 milliseconds and standard deviation of 0.2028 milliseconds. In experiments with data from ADAPT we also show that arithmetic circuit evaluation substantially outperforms joint tree propagation and variable elimination, two alternative algorithms for diagnosis using Bayesian network inference.

1 Introduction

In this work, we apply probabilistic model-based diagnosis techniques to a real-world electrical power system (EPS), namely the Advanced Diagnostic and Prognostic Testbed (ADAPT) [1]. A Bayesian network (BN) model of the ADAPT electrical power system plays a central role. This ADAPT BN represents health of sensors and subsystem components explicitly, and is auto-generated from a high-level system model of the ADAPT EPS. This BN is compiled, off-line, into an arithmetic circuit which is then evaluated on-line. We believe that this ADAPT case study clearly demonstrates how arithmetic circuits offer a scalable inference technique with potential for real-time evaluation in aircraft and spacecraft.

BNs have recently gained great popularity as an approach to representing multi-variate probability distributions [2]. BNs play a central role in a wide range of automated reasoning applications, for example in model-based diagnosis [3, 4, 5], medical diagnosis [6, 7], natural language understanding [8], probabilistic risk analysis [9], intelligent data analysis [10, 11], and error correction coding [12, 13].

A major point in this work is our case study on an electrical power system, ADAPT, that is representative of those found in aerospace vehicles. One of our main contributions is the integration of different techniques,

*Ole J. Mengshoel is with the Intelligent Systems Division and RIACS at the NASA Ames Research Center, Moffett Field, CA 94035 (phone: (650) 604-4199; email: Ole.J.Mengshoel@nasa.gov).

†Mark Chavira is affiliated with the Computer Science Department at the University of California, Los Angeles, CA 90095 (phone: (310) 825-7564; chavira@cs.ucla.edu). He is currently at Google.

‡Keith Cascio is with the Computer Science Department at the University of California, Los Angeles, CA 90095 (phone: (310) 825-7564; email: keith@cs.ucla.edu).

§Scott Poll is with the Intelligent Systems Division at the NASA Ames Research Center, Moffett Field, CA 94035 (phone: (650) 604-2143; email: Scott.Poll@nasa.gov).

¶Adnan Darwiche is with the Computer Science Department at the University of California, Los Angeles, CA 90095 (phone: (310) 206-5201; email: darwiche@cs.ucla.edu).

‖Serdar Uckun is with the Embedded Reasoning Area at the Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304 (email: Serdar.Uckun@parc.com). This work was performed while he was at NASA.

both existing and novel, in order to address a real world problem, thereby obtaining an approach that scales up to handle real world challenges in probabilistic model-based diagnosis.

Several aspects of this work make it different from previous efforts that utilize Bayesian networks for EPS diagnosis [4, 5]: A first contribution is our expression of EPS components and structure, using a novel high-level language, coupled with auto-generation of Bayesian networks from models expressed in this language. This approach supports the iterative development of probabilistic diagnostic models for large EPSs, including diagnostic system models that would be extremely tedious to hand-construct even for Bayesian network experts. The benefit of this approach to developers and engineers that are not, or only vaguely, familiar with Bayesian network appears to be even greater.

It is important to achieve real-time performance in many EPS health monitoring applications in aerospace [14, 15]. As a second contribution, we would like to highlight our compilation approach to probabilistic diagnosis, specifically the off-line compilation of Bayesian networks to arithmetic circuits [16, 17], which are then used for on-line diagnosis. In experiments, we have here shown that this approach provides high-quality diagnostic results on ADAPT scenarios. In addition, we have shown that performance is substantially better than alternative probabilistic inference algorithms, specifically variable elimination and clique (or join) tree propagation.

We believe that this work is significant for the following reasons. Electrical power systems are of crucial importance in aerospace as well as in numerous other areas of society [18, 1]. Our results in this work provide an argument for the feasibility of probabilistic, model-based diagnosis of EPSs. In particular, we address two real-world challenges of the model-based approach, especially as systems grow, namely (i) the difficulty of model construction (the modelling challenge) and (ii) scalability or performance concerns due to the inherent difficulty of probabilistic reasoning including diagnosis [19, 20, 21] (the real-time reasoning challenge). In the area of model construction, we provide automated generation of a Bayesian network from an EPS schematic and automated construction of an arithmetic circuit from the Bayesian network. This arithmetic circuit, which typically is large but has simple a semantics, supports real-time diagnosis on-line in the following two ways. First, it results in more predictable times. Second, it results in much faster inference. These two benefits are important to us, given the real-time requirements of aircraft and spacecraft avionics [15]. More generally, our approach to integrating a diagnostic system into the real-time setting is an example of "embedding AI into a real-time system" [14]. This addresses the scalability concerns mentioned above.

The rest of this work is structured as follows. After briefly introducing fundamentals of Bayesian networks and arithmetic circuits in Section 2, we discuss electrical power systems and ADAPT in particular in Section 3. We then present, using a simple EPS example, the high level specification language and model in Section 4. In Section 5 we consider the high level specification of a large portion of ADAPT. We then discuss Bayesian network modelling and auto-generation (Section 6 and Section 7 respectively), and compilation to arithmetic circuits (Section 8). We note that the latter two types of models (Bayesian networks and arithmetic circuits) are both auto-generated in our approach, thus reducing the amount of manual effort required. Finally, we report on experimental results in Section 9, both on real-world and synthetic data, before concluding in Section 10.

2 Preliminaries

We now briefly present the underlying formalisms of our probabilistic model-based reasoning approach: Bayesian networks and arithmetic circuits.

2.1 Bayesian Networks

Bayesian networks (BNs) represent multivariate probability distributions and are used for reasoning and learning under uncertainty [2]. Probability theory and graph theory form the basis of BNs: Roughly speaking, random variables are represented as nodes in a directed acyclic graph (DAG), while conditional dependencies are represented as graph edges. A key point is that a BN, whose graph structure often reflects a domain's causal structure, is a compact representation of a joint probability table if its graph is relatively sparse. Both discrete and continuous random variables can be represented in BNs; our main emphasis in this work is on BNs with discrete random variables. Each discrete random variable (or node) X has a finite number of states $\{x_1, \ldots x_m\}$ and is parameterized by a conditional probability table (CPT).

Let $\boldsymbol{X}$ be the BN nodes, $\boldsymbol{E} \subset \boldsymbol{X}$ the evidence nodes, and $\boldsymbol{e}$ the evidence. Different probabilistic queries can now be formulated (and supported by different algorithms as we will further discuss below). These queries assume that all nodes in $\boldsymbol{E}$ are clamped to values $\boldsymbol{e}$. Computation of most probable explanation

(MPE) amounts to finding a most probable explanation over the remaining nodes $\boldsymbol{R} = \boldsymbol{X} - \boldsymbol{E}$, or MPE($\boldsymbol{e}$). Computation of marginals (or beliefs) amounts to inferring the posterior probabilities over one or more query nodes $\boldsymbol{Q} \subseteq \boldsymbol{R}$, specifically BEL($Q, \boldsymbol{e}$) where $Q \in \boldsymbol{Q}$. Marginals are used to compute most likely values (MLVs) simply by picking, in BEL($Q, \boldsymbol{e}$), a most likely state. Computation of the maximum a posteriori probability (MAP) generalizes MPE computation and finds a most probable instantiation over nodes $\boldsymbol{Q} \subseteq \boldsymbol{R}$, MAP($\boldsymbol{Q}, \boldsymbol{e}$). Finally, we note that MAP can be approximated using MPE and MLV, and we will denote this using $\text{MAP}_{\text{MPE}}(\boldsymbol{Q}, \boldsymbol{e})$ and $\text{MAP}_{\text{MLV}}(\boldsymbol{Q}, \boldsymbol{e})$ respectively. $\text{MAP}_{\text{MPE}}(\boldsymbol{Q}, \boldsymbol{e})$ is the result of disregarding the nodes in $\boldsymbol{R}$ not in $\boldsymbol{Q}$, and $\text{MAP}_{\text{MLV}}(\boldsymbol{Q}, \boldsymbol{e})$ is the result of aggregating MLV($Q, \boldsymbol{e}$) of all $Q \in \boldsymbol{Q}$. These two approximations are of interest because of the greater computational complexity of MAP [21] compared to MPE and marginals [19, 20]. .

Different BN inference algorithms can be used to perform the above computations. We distinguish between exact and inexact algorithms. Exact algorithms include join tree propagation [22, 23, 24, 25], conditioning [26, 27], variable elimination [28, 29, 30], and arithmetic circuit evaluation [16, 17]. Inexact algorithms, and in particular stochastic local search algorithms, have been used to compute MPEs [31, 32, 33, 34, 35] as well as MAPs [36, 21] in Bayesian networks.

In resource-bounded systems, including real-time avionics systems, there is a strong need to align the resource consumption of diagnostic computation with resource bounds [14, 15]. Consequently, the compilation approach — join tree propagation and arithmetic circuit evaluation — is attractive in resource-bounded systems. The main advantage of compilation is that a significant amount of the work required for inference is performed once offline; this effort is then amortized over many online queries. Online inference is typically much faster when using a compilation approach, and the inference times often have much smaller variance. In this work we emphasize compilation into arithmetic circuits, which we present next.

2.2 Arithmetic Circuits

Arithmetic circuits (ACs), as discussed in [37, 16], are here used to perform probabilistic inference. The compilation from BNs to ACs is based on the following connection between BNs and multi-linear functions. With each Bayesian network, we associate a corresponding multi-linear function (MLF) that computes the probability of evidence. For example, the network in Figure 1—in which variables A and B are Boolean and C has three values—induces the following MLF:

$$\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\theta_{a_1}\theta_{b_1}\theta_{c_1|a_1,b_1} + \lambda_{a_1}\lambda_{b_1}\lambda_{c_2}\theta_{a_1}\theta_{b_1}\theta_{c_2|a_1,b_1} + \cdots + \lambda_{a_2}\lambda_{b_2}\lambda_{c_2}\theta_{a_2}\theta_{b_2}\theta_{c_2|a_2,b_2} + \lambda_{a_2}\lambda_{b_2}\lambda_{c_3}\theta_{a_2}\theta_{b_2}\theta_{c_3|a_2,b_2}.$$

The terms in the MLF are in one-to-one correspondence with the rows of the network's joint distribution. Assume that all *indicator variables* λ_x have value 1 and all *parameter variables* $\theta_{x|\boldsymbol{u}}$ have value $\mathsf{Pr}(x|\boldsymbol{u})$. Each term will then be a product of probabilities which evaluates to the probability of the corresponding row from the joint. The MLF will add all probabilities from the joint, for a sum of 1.0. To compute the probability $\mathsf{Pr}(\boldsymbol{e})$ of evidence $\boldsymbol{e}$, we need a way to exclude certain terms from the sum. This removal of terms is accomplished by carefully setting certain indicators to 0 instead of 1, according to the evidence.

The fact that a network's MLF computes the probability of evidence is interesting, but the network MLF has exponential size. However, if we can factor the MLF into something small enough to fit within memory, then we can compute $\Pr(\boldsymbol{e})$ in time that is linear in the size of the factorization. The factorization will take the form of an AC, which is a rooted DAG, where an internal node represents the sum or product of its children, and a leaf represents a constant or variable. In this context, those variables will be indicator and parameter variables. An example AC is depicted in Figure 1. We refer to this process of producing an AC from a Bayesian network as *compiling* the network. While a BN is more compact than an AC, for example as seen in Figure 1, there are in fact several advantages associated with using an AC for probabilistic inference, as we will discuss shortly.

Once we have an AC for a network, we can compute $\Pr(\boldsymbol{e})$ for given evidence $\boldsymbol{e}$ by assigning appropriate values to leaves and then computing a value for each internal node in bottom-up fashion. The value for the root is then the answer to the query. We can also compute answers to many other queries (a posterior marginal for each network variable, a posterior marginal for each network family, etc.) by performing a second downward pass [16] analogous to the outward pass of the jointree algorithm. Hence, many queries can be computed simultaneously in time linear in the size of the AC. MPE($\boldsymbol{e}$) may be computed in a similar manner, by using maximization nodes instead of addition nodes in the AC. Another main point is that the upward and downward passes may then be repeated for as many evidence sets as desired, without

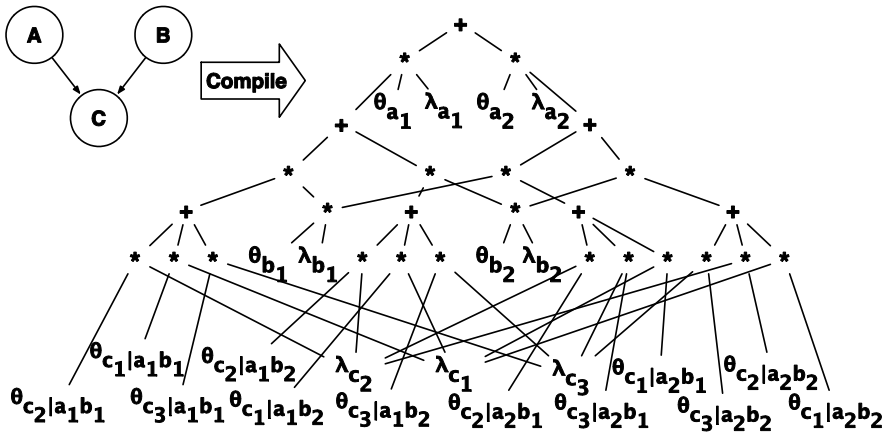

Figure 1: A Bayesian network (with nodes A, B, and C) and a corresponding arithmetic circuit compiled from the Bayesian network.

recompiling. Performing inference using an AC is therefore divided into two phases, an offline phase, which compiles the network into an AC and is run once, and an online phase, which answers many queries each time it is invoked, and which may be invoked multiple times.

We close this section by noting the close relationship between the jointree algorithm [22, 24] and ACs, since the data structures involved in this algorithm embed an AC in a very precise sense [38]. Other compilation algorithms have been developed based on tabular elimination [37], weighted model counting [39], and ADD elimination [17]. These algorithms can have an exponential advantage over jointree by exploiting structure in the parameters of the Bayesian network [40].

3 Electrical Power Systems

We discuss the importance of electrical power systems (EPSs) in aerospace applications and describe an EPS testbed, ADAPT, that is the subject of this case study.

3.1 Electrical Power System Challenges in Aerospace and at NASA

Electrical power systems (EPSs) play an essential role in aerospace vehicles [18, 1]. The electrical power system may be thought of as the circulatory system of an aerospace vehicle. In the human body, the circulatory system delivers the necessary nutrients to the constituent elements. Similarly, the EPS delivers energy to subsystems in order to power required vehicle functions such as life support, propulsion, communications, guidance, navigation, and control. Loss of electrical power to these and similar subsystems can result in severe repercussions for the vehicle, personnel, or mission.

Unfortunately, electrical power systems have been implicated in several aerospace vehicle incidents, accidents and mishaps. In one accident, the left Power Conversion and Distribution Unit (PCDU) on a Boeing 717 failed, resulting in the loss of the left AC and DC busses. The most likely cause was determined to be the failure of a transient suppression diode, which allowed AC current to contaminate the DC circuits of the PCDU. Shorted diodes were found on one of the circuit boards and several circuit cards showed evidence of heavy sooting. One serious and several minor injuries were sustained by passengers using the emergency evacuation slides after the pilot ordered the crew and passengers to evacuate the airplane subsequent to smelling electrical smoke (see NTSB report number NYC03FA067). In another incident involving the PCDU of a Boeing 717, a tantalum capacitor and a permanent magnet generator input transformer failed, resulting in smoke in the cabin and an emergency landing and evacuation (NTSB report ATL04IA085).

The Electric Propulsion Space Experiment (ESEX) mission, launched and operated in early 1999, ended prematurely when the spacecraft experienced a catastrophic battery failure. During the first battery charge, a charging circuit instability occurred due to high internal impedance. After the eighth firing, large battery voltage fluctuations were observed over a 24 hour period, eventually stabilizing at a low voltage. The failure was most likely the result of electrolyte leakage which caused a short circuit to the battery case. Subsequent

Figure 2: The ADAPT electrical power system lab at the NASA Ames Research Center.

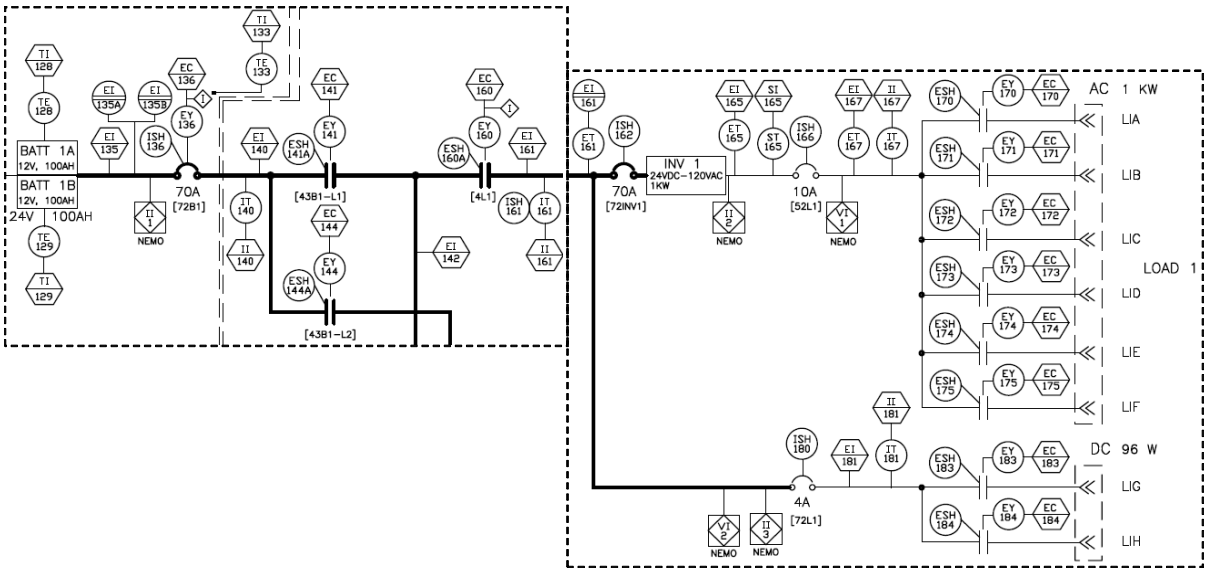

Figure 3: Schematic of a portion of the ADAPT testbed, showing one of three power storage parts (for Battery 1, left) and one of two load banks (Load bank 1, right). Detailed information about loads is given in Table 2. Figure 4 shows pictures of loads as well as how they are connected to load banks.

discharging through the short circuit led to increases in battery temperature and pressure due to gasses released from electrolyte decomposition. This resulted in a breach of the battery case, entry of super-heated gas into the flight unit, and eventual venting into space [41]. On January 14 2005, an Intelsat operated communications satellite suffered a total loss after a sudden and unexpected electrical power system anomaly. The failure of Intelsat 804's high voltage power system was likely the result of a high current event in the battery circuitry triggered by an electrostatic discharge (see `http://sat-nd.com/failures/index.html/`). A battery failure also occurred on the Mars Global Surveyor, which last communicated with Earth on November 2, 2006. In this case, a software error oriented the spacecraft to an angle that exposed it to too much sunlight. This caused the battery to overheat and ultimately led to the depletion of both batteries (see `http://mpfwww.jpl.nasa.gov/mgs/newsroom/20070413a.html`). New challenges will be presented by advanced battery chemistries which require careful supervision.

These are just a few examples of the faults that can arise in EPSs. In addition to failures in PCDUs and batteries, failures in wiring, electronics and electromechanical devices are also well-known. Much attention has recently been given to wiring in commercial aircraft, with the cancellation of hundreds of flights because of concerns about the bundling and routing of wires. Power semiconductor electronics and electromechanical devices will play an ever greater role as aerospace vehicles move towards all electric designs. Given the prevalence and importance of EPSs, it is vital to develop effective health management techniques for real-time aerospace operations, including at NASA.

3.2 ADAPT: Advanced Diagnostic and Prognostic Testbed

Figure 2 shows the Advanced Diagnostics and Prognostics Testbed (ADAPT) lab; see also `http://ti.arc.nasa.gov/adapt/` and [1]. ADAPT, which has capabilities for power generation, power storage, and power distribution, is a fully operational electrical power system that is representative of such systems in aircraft and spacecraft. Figure 3 presents a schematic with a representative battery and load bank from ADAPT. Figure 4 shows ADAPT loads. ADAPT is configured to achieve fault-tolerance, and contains three batteries and two load banks. One battery can provide power to at most one load bank at any given time, and each battery is wired such that it can power any one of the two load banks. In Figure 3, for example, for Battery 1 to power Load bank 1, relay EY141 must be closed. For Battery 1 to power Load bank 2, on the other hand, relay EY144 must be closed. Relays EY141 and EY144 cannot both be closed at the same time.

Different types of components and sensors used in ADAPT are presented in Table 1. Relays, which are commanded to close and open in order to control power, have prefix EY (in Figure 3) and health modes healthy, stuck or failed open, and stuck or failed closed. A position sensor, also presented in Table 1, reports

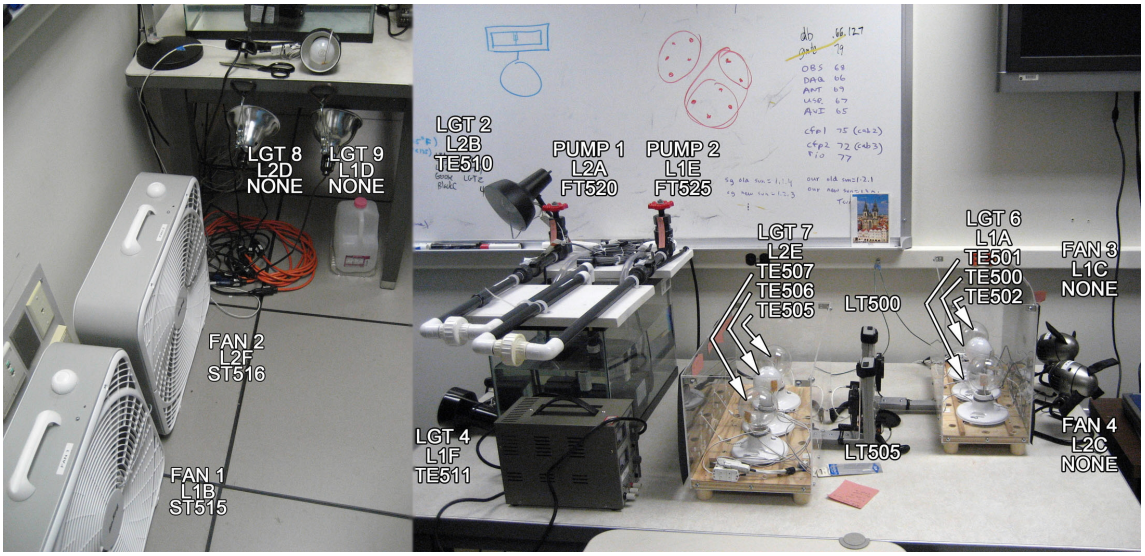

Figure 4: The loads in ADAPT along with their sensor and connection information. For example, consider the large fan FAN1 shown at the bottom left in the picture. FAN1's labels say that the fan is connected to L1B in the schematic (see Figure 3) and has one sensor ST515 (see Table 2).

on the status of a relay. As concrete examples, consider in Figure 3 relay EY170 that controls power to load L1A; it also has a position (or touch) sensor ESH170. In our application, each of EY170 and ESH170 are represented by random variables including health status random variables with states as represented in Table 1. For example, EY170's health random variable has states *{healthy, stuckOpen, stuckClosed}*. Upstream of relay EY170 is a current sensor IT167; the states of its health variable are *{healthy, readCurrentLo, readCurrentHi}* as shown in the table. Further information regarding our probabilistic modelling of EPS components and structure is provided in Section 6 and Section 7.

There are two load banks in ADAPT, each has an AC part and a DC part. Load bank 2 is very similar to Load bank 1, the loads are just plugged into different locations. Each load is connected at a fixed place in the power distribution unit. In other words, for the purposes of this work there is no ambiguity as to which "power outlet" a load is "plugged into". At this time, there are mostly AC loads in ADAPT; see Table 2 as well as Figure 4. We have included the possibility of DC loads (currently there are 2 DC loads, one for each load bank). To convert DC power from the batteries into AC power used by the AC loads, ADAPT has two inverters, one per load bank. A failed inverter breaks power transmission to the AC loads; see the *stuckOpen* failure mode in Table 1.

4 High Level Models

Our approach to probabilistic model–based diagnosis involves four stages. In the first stage, we describe the EPS using a high–level modeling language. The main purpose of this language is to make specifying the EPS easy and less error–prone. In the second stage, we apply a program to automatically convert the high–level specification into a Bayesian network. Putting the model into the form of a Bayesian network allows us to leverage a large body of existing work on Bayesian network inference techniques. In the third stage, we compile the Bayesian network into an arithmetic circuit. This stage represents the application of a specific technique (arithmetic circuits) for performing inference in Bayesian networks, which in the resource-bounded, real-time context has significant advantages over other techniques [14, 15]. All stages up to this point have taken place offline, before the EPS is put into actual use. The fourth stage involves applying algorithms to the arithmetic circuit to perform inference online, when the EPS is deployed in an aircraft or spacecraft. By this time, as much computational effort as possible has been performed offline, leaving much less computation to be performed online. In this and the next sections, we provide more detail on each stage, beginning in this section with the novel high-level specification language.

A high-level specification is a sequence of statements. The syntax of our high–level specification language is given in Table 3. In Table 3, <name> is an identifier and <p> is a probability. Each statement defines

Part	Prefix	Mode (Healthy/Faulty)	States
Battery	BATT	Healthy	*healthy*
		Voltage failure or drain	*stuckDisabled*
Circuit breaker	ISH	Healthy	*healthy*
		Stuck or failed open	*stuckOpen*
		Stuck or failed closed	*stuckClosed*
Inverter	INV	Healthy	*healthy*
		Switched off	*stuckOpen*
Relay	EY	Healthy	*healthy*
		Stuck or failed open	*stuckOpen*
		Stuck or failed closed	*stuckClosed*
Voltage sensor	EI	Healthy	*healthy*
		Reading stuck low	*readVoltageLo*
		Reading stuck high	*readVoltageHi*
Current sensor	IT	Healthy	*healthy*
		Reading stuck low	*readCurrentLo*
		Reading stuck high	*readCurrentHi*
Position sensor	ISH	Healthy	*healthy*
		Reading stuck open	*stuckOpen*
		Reading stuck closed	*stuckClosed*

Table 1: Different EPS parts along with their modes and the corresponding states of the health node for the part.

ID	Type	Relay	Description	Load ID	Measurements (Sensor IDs)
L1A	AC	EY170	3 light bulbs	LGT6	Temperatures (TE500, TE501, TE502) and Light sensor (LT500)
L1B	AC	EY171	Big fan	FAN1	RPM (ST515)
L1C	AC	EY172	Small fan	FAN3	None
L1D	AC	EY173	1 light bulb	LGT8	None
L1E	AC	EY174	Water pump	PMP2	Flow rate (FT525)
L1F	AC	EY175	1 light bulb	LGT4	Temperature (TE511)
L1G	DC	EY183	Electromechanical	DC1	None
L1H	DC	EY184	None	N/A	N/A

Table 2: Loads and their sensors (where applicable) for Load bank 1 of the ADAPT electrical power system.

a component, which can either be a *source* (battery), a *basic* component, a *sensor*, or a *sink* (load). For brevity, we do not describe here some statements defining more complicated sensors. The general idea is that electricity flows from sources through basic components to sinks. Along the way, sensors attempt to monitor what is happening, and various failures can occur at each component. For each component, we define its name, its type (e.g., `source`, `load`, `breaker`, `relay`, `sensorCurrent`, `sensorVoltage`), the probability that the component will fail,[1] and a set of neighboring components. For a source, the set of neighbors is empty; for a basic component or a sink, we list all upstream neighbors, neighbors that lie between the component and a source of electricity; for a sensor, we list only the component to which the sensor is attached. These sets of neighbors serve to define the topology of an EPS; in fact a directed acyclic graph is induced as discussed in Section 7.

Figure 5 depicts a very simple example of an electrical power system, in which there is only one battery (source) and only one load (sink). Electricity flows from the battery `batt` to the load `ld` through a relay `rly`. There is a current sensor `currSens` attached to the wire `wire1` between the source and the relay and a touch sensor attached to the relay itself. There is also a voltage sensor `voltSens` attached to the second wire, `wire2`. Table 4 contains a description of this small EPS in our specification language. The third line, for example, defines a current sensor `curSens` with failure probability 0.0003 and attached to component `wire1` (which happens to be a wire defined in the second line); the fourth line defines a relay `rly` with failure

[1] As described, all failures for a given component have equal probability, but the syntax can easily be extended to assign differing probabilities to different kinds of failures.

```
<eps> ::= <component>+
<component> ::= (<source> | <basic> | <sensor> | <sink>) ";"
<source> ::= <name> ":" "source" ":" <p> ":"
<basic> ::= <name> ":" <btype> ":" <p> ":" <name>+
<sensor> ::= <name> ":" <stype> ":" <p> ":" <name>
<sink> ::= <name> ":" "sink" ":" <p> ":" <name>+
<btype> ::= "load" | "wire" | "inverter" | "breaker" | "relay"
<stype> ::= "sensorCurrent" | "sensorVoltage" | "sensorTouch"
```

Table 3: The syntax of our novel specification language for electrical power systems.

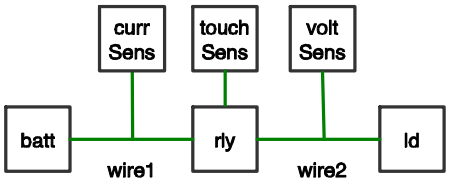

Figure 5: A small electrical power system; it is described using our specification language in Table 4.

probability 0.0003 and receiving electricity from component `wire1`; and the fifth line defines a touch (or position) sensor `touchSens` with failure probability 0.0002 and attached to component `rly`.

The main advantage of the high–level specification is ease-of-use. One can specify a model by listing which components exist in the system, and for each, its type, failure probability, and neighbors. All of this information can be obtained directly from schematics and hardware manuals. Consequently, the modeling task at the specification language level does not require guesswork or any knowledge of Bayesian networks or arithmetic circuits.

Components differ from each other in some ways that are not represented explicitly in the specification language, because the information can be inferred from the component's type. For example, some component types, such as a circuit–breaker, accept a command to open or close, whereas some, such as a wire, do not. Similarly, different components may suffer different types of failures as presented in Table 1. For example, a wire can only fail in a stuck–open state, whereas a circuit breaker can be stuck–open or stuck–closed.[2] This information is added during the BN auto-generation stage (see Section 7), reflecting our BN modelling approach, which is what we discuss next.

5 High Level Model for ADAPT

Using our specification language, the ADAPT EPS is described using statements for the following components: 3 sources (batteries), 20 sinks (loads), 16 wires (we only need to describe a wire if it has a sensor attached or if we want to model failures in wires), 2 inverters, 9 circuit breakers, 25 relays, 17 current and

[2]Sometimes a subtle distinction between "fail" and "stuck" is made, which we currently do not make in the BN; instead we for simplicity call all failures "stuck". (For example, "failed open" means that the component was closed and then opened without any command. "Stuck open", on the other hand, means that the component was already open, and failed to close when commanded to do so.)

batt :	source :	0.0001 ;	
wire1 :	wire :	0.0000 :	batt ;
curSens :	sensorCurrent :	0.0003 :	wire1 ;
rly :	relay :	0.0003 :	wire1 ;
touchSens :	sensorTouch :	0.0002 :	rly ;
wire2 :	wire :	0.0000 :	rly ;
voltSens :	sensorVoltage :	0.0002 :	wire2 ;
ld :	sink :	0.0001 :	wire2 ;

Table 4: A small EPS, shown in Figure 5, described using our specification language.

load sensors, 16 voltage sensors, 33 position (touch) sensors, and 6 more advanced sensors (these advanced sensors are not described here for sake of brevity). We now discuss the high level model of ADAPT in some detail. In particular, we present the part that models the ADAPT schematic shown in Figure 3. ADAPT loads are depicted in Figure 4.

5.1 Modelling of Batteries and Distribution Network

The specification language follows the components and structure of an electrical power circuit very closely, hence models like the ADAPT model we discuss next can be read directly off schematics. ADAPT contains three batteries and distribution networks. Due to their similarities, we only present the specification for one of them (Battery 1) here; see also the schematic in Figure 3:

```
battery1 :  battery :  0.0005;
wire135 :  wire :  0.0000 :  battery1;
e135 :  sensorVoltage :  0.0005 :  wire135;
breaker_ey136_op :  breaker :  0.0005 :  wire135;
ish136 :  sensorTouch :  0.0005 :  breaker_ey136_op;
wire140 :  wire :  0.0000 :  breaker_ey136_op;
e140 :  sensorVoltage :  0.0005 :  wire140;
it140 :  sensorCurrent :  0.0005 :  wire140;
relay_ey141_cl :  relay :  0.0005 :  wire140;
esh141a :  sensorTouch :  0.0005 :  relay_ey141_cl;
relay_ey144_cl :  relay :  0.0005 :  wire140;
esh144a :  sensorTouch :  0.0005 :  relay_ey144_cl;
```

We now consider the part that connects Battery 1 with Load bank 1:

```
wire142 :  wire :  0.0000 :  relay_ey141_cl relay_ey241_cl relay_ey341_cl;
e142 :  sensorVoltage :  0.0005 :  wire142;
relay_ey160_cl :  relay :  0.0005 :  wire142;
esh160a :  sensorTouch :  0.0005 :  relay_ey160_cl;
wire161 :  wire :  0.0000 :  relay_ey160_cl;
e161 :  sensorVoltage :  0.0005 :  wire161;
it161 :  sensorCurrent :  0.0005 :  wire161;
```

5.2 Modelling of Load Banks and Loads

ADAPT contains two load banks with loads. Due to the similarities between loads, we only discuss the specification of loads L1A, L1B, L1C, L1E, and L1G for Load bank 1 here. First, we present the part of the specification that concerns the branch that leads to and includes the DC load L1G:

```
breaker_ey180_op :  breaker :  0.0005 :  wire161;
ish180 :  sensorTouch :  0.0005 :  breaker_ey180_op;
wire181 :  wire :  0.0000 :  breaker_ey180_op;
e181 :  sensorVoltage :  0.0005 :  wire181;
it181 :  sensorCurrent :  0.0005 :  wire181;
relay_ey183_cl :  relay :  0.0005 :  wire181;
esh183 :  sensorTouch :  0.0005 :  relay_ey183_cl;
load183 :  sink :  0.0005 :  relay_ey183_cl;
```

What follows below is the branch containing the Load bank 1 inverter:

```
breaker_ey162_op :  breaker :  0.0005 :  wire161;
ish162 :  sensorTouch :  0.0005 :  breaker_ey162_op;
inv1 :  inverter :  0.0005 :  breaker_ey162_op;
wire165 :  wire :  0.0000 :  inv1;
e165 :  sensorVoltage :  0.0005 :  wire165;
breaker_ey166_op :  breaker :  0.0005 :  wire165;
ish166 :  sensorTouch :  0.0005 :  breaker_ey166_op;
wire167 :  wire :  0.0000 :  breaker_ey166_op;
e167 :  sensorVoltage :  0.0005 :  wire167;
it167 :  sensorCurrent :  0.0005 :  wire167;
```

Here are the AC loads L1B, L1C, and L1G:

```
relay_ey171_cl : relay : 0.0005 : wire167;
esh171 : sensorTouch : 0.0005 : relay_ey171_cl;
load171 : sink : 0.0005 : relay_ey171_cl;
fan1_speed_st515 : sensorCurrentThree : 0.0005 : load171;
relay_ey172_cl : relay : 0.0005 : wire167;
load172 : sink : 0.0005 : relay_ey172_cl;
esh172 : sensorTouch : 0.0005 : relay_ey172_cl;
load174 : sink : 0.0005 : relay_ey174_cl;
pump2_flow_ft525 : sensorCurrentThree : 0.0005 : load174;
relay_ey174_cl : relay : 0.0005 : wire167;
esh174 : sensorTouch : 0.0005 : relay_ey174_cl;
relay_ey170_cl : relay : 0.0005 : wire167;
```

We finally consider AC load L1A.

```
esh170 : sensorTouch : 0.0005 : relay_ey170_cl;
load170_bulb0 : sink : 0.0005 : relay_ey170_cl;
box1_light1_temp_te500 : sensorCurrent : 0.0005 : load170_bulb0;
load170_bulb1 : sink : 0.0005 : relay_ey170_cl;
box1_light2_temp_te501 : sensorCurrent : 0.0005 : load170_bulb1;
load170_bulb2 : sink : 0.0005 : relay_ey170_cl;
box1_light3_temp_te502 : sensorCurrent : 0.0005 : load170_bulb2;
light1_level_lt500 : sensorCurrentCount : 0.0005 : load170_bulb0 load170_bulb1 load170_bulb2;
```

There are here three light bulbs, and this is reflected as three loads in the specification, one for each light bulb: `load170_bulb0`, `load170_bulb1`, and `load170_bulb2`. There are two types of sensors: (i) A temperature sensor, which is glued to a light bulb. There is a "current" sensor for each of these: `box1_light1_temp_te500`, `box1_light2_temp_te501`, and `box1_light3_temp_te502`. Clearly, a current sensor is not the same as a temperature sensor technically, but they both really measure current, and by using different failure probabilities we account for their differences. (ii) A light sensor, which is a few inches away from all the light bulbs. This light sensor senses three light bulbs. This is represented by another "current" sensor, `light1_level_lt500`, attached to all three bulbs. What is different from the temperature sensors, for example, is that this sensor is attached multiple components and reports the number of components that were on. Also, there is just a single failure state. In the specification language, we note how a sensor is attached to components. Sensors are specified very similarly to how normal components are specified.

6 Modelling Electrical Power Systems using Bayesian Networks

A main contribution in this work is our systematic modelling of EPSs using BNs. BNs provide a probabilistic semantics for our high-level specification language, and in addition they support efficient inference including compilation into arithmetic circuits. In order to discuss our modelling approach as well as the supporting auto-generation algorithm and software, we partition the set of BN nodes $\boldsymbol{X}$ into subsets $\boldsymbol{H}$, $\boldsymbol{E}$, $\boldsymbol{C}$, $\boldsymbol{P}$ and $\boldsymbol{R}$ as follows:

- Health nodes ($\boldsymbol{H}$), where $\boldsymbol{H} = \boldsymbol{H}_C \cup \boldsymbol{H}_S$ and $\boldsymbol{H}_C \cap \boldsymbol{H}_S = \varnothing$. Here, $\boldsymbol{H}_C$ (component health nodes) represent health of the EPS components and $\boldsymbol{H}_S$ (sensor health nodes) represent the health of the EPS sensors.
- Evidence nodes ($\boldsymbol{E}$), where $\boldsymbol{E} = \boldsymbol{E}_C \cup \boldsymbol{E}_S$ and $\boldsymbol{E}_C \cap \boldsymbol{E}_S = \varnothing$. Here, $\boldsymbol{E}_C$ (command nodes) represent the commands to the EPS, while $\boldsymbol{E}_S$ (sensor nodes) represent sensor readings from the EPS.
- Connection nodes ($\boldsymbol{C}$), where $\boldsymbol{C} = \boldsymbol{C}_U \cup \boldsymbol{C}_S$ and $\boldsymbol{C}_U \cap \boldsymbol{C}_S = \varnothing$. Here, $\boldsymbol{C}_U$ (source connection nodes) represent connection to a source (battery) in an EPS; $\boldsymbol{C}_S$ (sink connection nodes) represent connection to a sink (load) in an EPS.
- Presence nodes ($\boldsymbol{P}$), where $\boldsymbol{P} = \boldsymbol{P}_C \cup \boldsymbol{P}_V$ and $\boldsymbol{P}_C \cap \boldsymbol{P}_V = \varnothing$. Here, $\boldsymbol{P}_V$ (voltage presence nodes) represent voltage, similar to water pressure, provided by a source (battery) in an EPS. $\boldsymbol{P}_C$ (current propagation or presence nodes) represent flow, similar to water flow, of electrical current from a source (battery) to a sink (load) in an EPS. In our case, there is presence of voltage iff there is a closed connection to one or more batteries, therefore one may work with either $\boldsymbol{C}_U$ or $\boldsymbol{P}_V$.

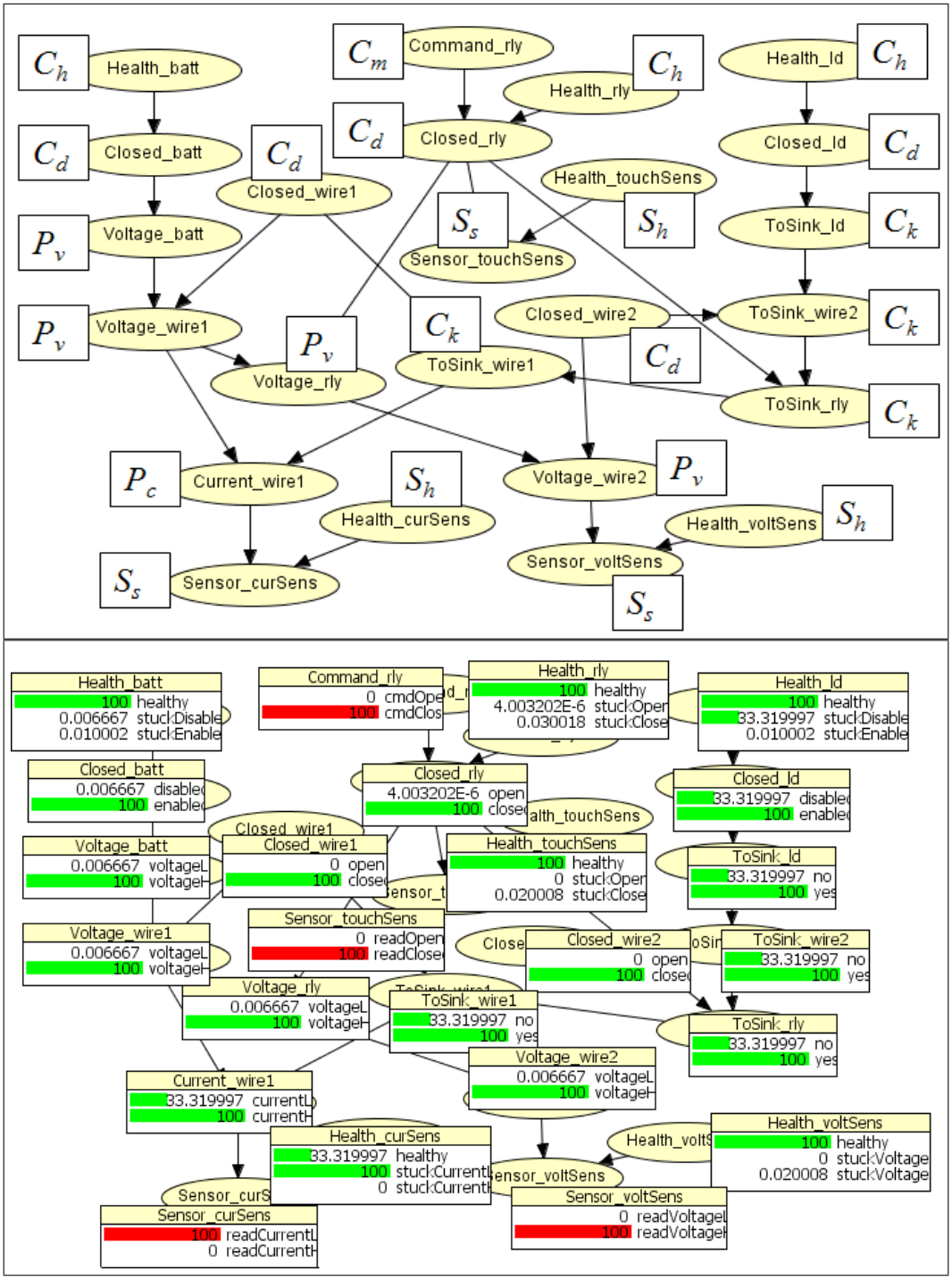

Table 5: Bayesian network (top) and MPE computation using the Bayesian network (bottom) for the small EPS specified in Table 4. In the Bayesian network we show both the actual BN node names (*Health_batt*, *Health_ld*, ...) as well as the mathematical notation (C_h, C_d, ...) used to describe the auto-generation algorithm.

- Remaining EPS nodes ($\boldsymbol{R}$): Nodes that are not health, evidence, connection, or presence nodes. If $\boldsymbol{X}$ is the set of all BN nodes, then $\boldsymbol{R} = \boldsymbol{X} - \boldsymbol{H} - \boldsymbol{E} - \boldsymbol{C} - \boldsymbol{P}$.

The above node partitioning is important for several reasons. It allows us to: state different probabilistic queries of interest; discuss our EPS modelling approach using BNs (both the topology as well as the individual nodes associated with different EPS components); and clearly present the experimental protocol.

In Section 2 we discussed, given query variables $\boldsymbol{Q} \subseteq \boldsymbol{X}$ and evidence $\boldsymbol{e}$, three probabilistic queries: $\text{MAP}(\boldsymbol{Q}, \boldsymbol{e})$, $\text{MAP}_{\text{MPE}}(\boldsymbol{Q}, \boldsymbol{e})$, and $\text{MAP}_{\text{MLV}}(\boldsymbol{Q}, \boldsymbol{e})$. By introducing the above partitioning, we can put $\boldsymbol{Q} = \boldsymbol{H}_C$, $\boldsymbol{Q} = \boldsymbol{H}_S$, or $\boldsymbol{Q} = \boldsymbol{H}$ and obtain a total of nine different diagnostics queries. As an example, $\boldsymbol{Q} = \boldsymbol{H}_S$ is of interest in sensor validation, where the main focus is on qualifying and disqualifying sensors [42], for instance voltage sensors, current sensors, fuel sensors, or altitude sensors. In the rest of this work we emphasize, in the interest of brevity, only $\boldsymbol{Q} = \boldsymbol{H}$ and in particular $\text{MAP}_{\text{MPE}}(\boldsymbol{H}, \boldsymbol{e})$.

A key contribution in this work is our modeling of EPSs using Bayesian networks. An EPS presents two different but closely related problems, namely a voltage presence problem and a current flow problem. Voltage may propagate from a battery towards the loads. For current to flow, there must be voltage present and in addition the EPS circuit needs to be closed, which typically happens when an EPS load is turned on and all other relays between the load and a battery are also closed. This bidirectional voltage-current propagation problem is different from, and more complicated than, the unidirectional flow problem posed by digital circuits implementing boolean logic. Such digital electronic circuits have been extensively studied in the model-based diagnosis and Bayesian network literature [2].

Table 5 provides a simple example of our EPS modelling approach. This BN represents the simple electrical power system introduced in Figure 5 and in Table 4.[3] Here, $\boldsymbol{H}_C = \{$*Health_ batt*, *Health_ ld*$\}$ and $\boldsymbol{H}_S = \{$*Health_ curSens*, *Health_ voltSens*, *Health_ touchSens*$\}$; $\boldsymbol{E}_C = \{$*Command_ relay*$\}$ and $\boldsymbol{E}_S = \{$*Sensor_ curSens*, *Sensor_ voltSens*, *Sensor_ touchSens*$\}$. The topology of the ADAPT BN, which currently contains over 500 nodes, is analogous to this BN's topology. A key point in this example is how the integration of voltage presence nodes ($\boldsymbol{P}_V = \{$*Voltage_ batt*, *Voltage_ wire1*, *Voltage_ rly*, *Voltage_ wire2*$\}$), sink connection nodes ($\boldsymbol{C}_S = \{$*ToSink_ wire1*, *ToSink_ rly*, *ToSink_ wire2*, *ToSink_ ld*$\}$), and the current flow node ($\boldsymbol{P}_C = \{$*Current_ wire1*$\}$) help solve the bidirectional voltage-current propagation problem. Many nodes, including current flow nodes, can be pruned (and indeed have been here) because they are leaf nodes and not involved in sensors. Another key point is how sensors, for example the voltage sensor (nodes *Health_ voltSens*, *Sensor_ voltSens*) and the current sensor (nodes *Health_ curSens*, *Sensor_ curSens*), are integrated into the overall BN topology.

We now consider inference as illustrated in Table 5. Suppose that $\boldsymbol{e} = \{$*Command_ rly = cmdClose*, *Sensor_ curSens = readCurrentLo*, *Sensor_ voltSens = readVoltageHi*, *Sensor_ touchSens = readClosed*$\}$ (see Table 5). This gives $\text{MAP}_{\text{MPE}}(\boldsymbol{H}, \boldsymbol{e}) = \{$*Health_ batt = healthy*, *Health_ ld = healthy*, *Health_ curSens = stuckCurrentLo*, *Health_ voltSens = healthy*$\}$. In words, if the command and sensor readings, except for *Sensor_ curSens = readCurrentLo*, suggest that power is supplied to the load, then the MPE diagnosis is that all components and sensors are healthy, except for the current sensor, where *Health_ curSens = stuckCurrentLo*. It is reassuring that there is agreement between the MPE diagnosis and common sense.

While our modelling approach can be used when manually constructing BNs for EPSs, it is even more powerful when automated, and we now turn to how we have formalized it in the form of an auto-generation algorithm.

7 Auto-generation of Bayesian Networks

In this section, we discuss how a BN is auto-generated from a high-level specification model. This is the second stage in our approach to probabilistic model–based diagnosis. The conversion runs in a loop, which processes one component from the specification in each iteration. Before embarking on the discussion of the auto-generation algorithm, we note that there are fundamentally two ways to go about this: Either the components in the specification need to be traversed in a particular order, so that when the edges between a node and its parents are created, the parent nodes are guaranteed to already exist. Alternatively, we need to take two passes, where the first pass creates the nodes and the second pass adds the edges. For simplicity, we take the former approach here, and traverse the specification in a certain sequential order. Such a sequential order is guaranteed to exist under the assumption that the underlying EPS can be described using a directed acyclic graph (DAG). There is a clear mapping from a high-level specification to a DAG: In each component

[3] In fact, this BN was auto-generated, as discussed in Section 7, from the specification in Table 4.

statement (see Table 3), the first <name> represents a node, and the <name>+ part represents its parents (assuming this is not a source statement in the specification, which is trivial since it is a root node in the DAG). Under the assumption that there exists such a DAG, there exists a sequential high-level specification, since it is well-known from graph theory that any DAG can be topologically (or sequentially) sorted.

The auto-generation algorithm can now be summarized as follows: We iterate over the components in the specification and for each generate a set of BN nodes and a set of BN edges. Each time the algorithm creates a BN node for a component, it places the node into the appropriate set among $\boldsymbol{H}_C$, $\boldsymbol{H}_S$, $\boldsymbol{E}_C$, $\boldsymbol{E}_S$, $\boldsymbol{C}_U$, $\boldsymbol{C}_S$, $\boldsymbol{P}_C$, $\boldsymbol{P}_V$, and $\boldsymbol{R}$, as we illustrate below. Sensors are handled differently from other components, which are fundamentally handled all the same.

The processing of a sensor is somewhat different from the processing of other components, so we treat sensors separately, after first discussing other components. As an example, Figure 6(a) depicts the part of a BN corresponding to a relay C. For the component C, the auto-generation algorithm generates six nodes in the BN:

- $C_h \in \boldsymbol{H}_C$, with values {*healthy*, *stuckOpen*, *stuckClosed*}, indicates C's health state. C_h has a CPT set according to C's failure probability as defined in the high–level specification.
- $C_m \in \boldsymbol{E}_C$, with values {*cmdOpen*, *cmdClose*}, indicates the command being sent to the relay. This value will always be known prior to inference, since it is set according to the command being issued to the relay (typically, it is *cmdClose*). Therefore, probabilities in this CPT are not important; C_m has a uniform CPT.
- $C_d \in \boldsymbol{R}$, with values {*open*, *closed*}, indicates whether C is currently closed. If C_h = *healthy*, then C_d indicates closed iff C_m = *cmdClose*. Otherwise, if C_h is *stuckOpen* (*stuckClosed*), then C_d indicates *open* (*closed*).
- $C_r \in \boldsymbol{C}_U$, with values {*open*, *closed*}, indicates whether there is a closed path from C to a battery (source). C_r = *closed* iff C_d = *closed* $\wedge \vee_N [N_r$ = *closed*] where N iterates over all of C's upstream neighbors[4].
- $C_k \in \boldsymbol{C}_S$, with values {*open*, *closed*}, indicates whether there is a closed path from C to a load (sink). C_k = *closed* iff C_d = *closed* $\wedge \vee_N [N_k$ = *closed*] where N iterates over all of C's downstream neighbors.
- $C_c \in \boldsymbol{P}_C$, with values {*currentLo*, *currentHi*}, indicates whether current is flowing through C. C_c = *currentHi* iff C_r = *closed* and C_k = *closed*.

For C_r and C_k, the disjunction is cascaded to prevent the CPT from becoming too large. This same template applies to all non–sensor components with a few minor modifications. For example, a source can set C_r to be equivalent to C_d; a sink can set C_k to be equivalent to C_d; a wire, which does not accept commands, will always set C_m to *cmdClosed* (or omit C_m from the model); and different component types may have different types of failures.

Figure 6(b) depicts the part of the BN corresponding to a current sensor S, which is attached to a node such as C_c of a component C. The auto-generation algorithm creates two nodes in the BN corresponding to S :

- $S_h \in \boldsymbol{H}_S$, with values {*healthy*, *stuckCurrentLo*, *stuckCurrentHi*}, indicates S's health state. S_h has a CPT set according to S's failure probability as defined in the high–level specification.
- $S_s \in \boldsymbol{E}_S$, with values {*readCurrentLo*, *readCurrentHi*}, indicates S's two-state discretized sensor reading. If S_h = *healthy*, then S_s indicates closed iff C_c = *currentHi*. Otherwise, if S_h is *stuckCurrentLo* (*stuckCurrentHi*), then S_s indicates *readCurrentLo* (*readCurrentHi*).

This same template applies to all sensor components (except some more complicated sensors, which are beyond the scope of this work) with a few minor modifications. For example, different sensors are attached to different nodes in C: current sensors are attached to C_c, voltage sensors are attached to C_r, while touch sensors are attached to C_d.

After the BN generation step discussed above, there is a BN pruning step. Pruning takes place based on information about query nodes ($\boldsymbol{H}_C$ and $\boldsymbol{H}_S$) as well as about evidence nodes ($\boldsymbol{E}_C$ and $\boldsymbol{E}_S$). It is

[4] A neighbor of C is upstream of C if it is located between C and a source in the high-level specification. A neighbor of C is downstream of C if it is located between C and a sink in the high-level specification.

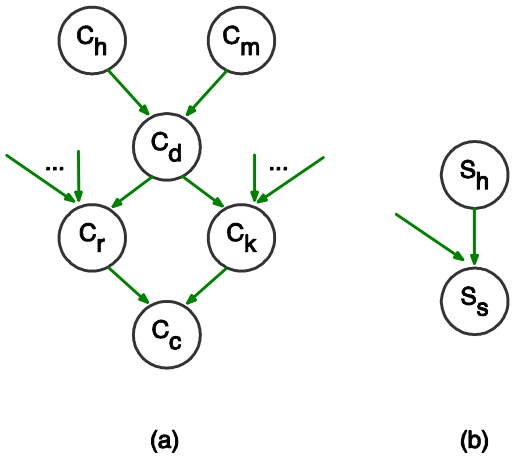

Figure 6: The part of the BN corresponding to (a) a relay and (b) a current sensor.

well-known that pruning can make inference in Bayesian networks more tractable. A common pruning technique involves removing leaf nodes that are not part of the evidence ($\boldsymbol{E}_C \cup \boldsymbol{E}_S$) or the query ($\boldsymbol{H}_C$, $\boldsymbol{H}_S$, or $\boldsymbol{H}_C \cup \boldsymbol{H}_S$) [43]. In Table 5, some of the nodes have been pruned compared to Figure 6(a). Specifically, nodes corresponding to C_r, C_k, and C_c are pruned in Table 5. In other words, for the relay shown in Table 5 we have the following correspondence with the non-pruned nodes in Figure 6(a): C_h = *Health_ rly*, C_m = *Command_ rly*, C_d = *Closed_ rly*. How can we determine to prune C_r, C_k, and C_c (referring to Figure 6(a)), but not prune S_s = *Sensor_ curSens* and S_h = *Health_ curSens* (referring to Figure 6(b) and Table 5)? Here, S_s = *Sensor_ curSens* is a variable for which we assert evidence, while S_h = *Health_ curSens* is a query variable. Evidence and query variables are never pruned. We only prune non-evidence, non-query variables that are leaves, or which become leaves as a result of other pruning. Consequently, all of C's nodes are pruned except the following: C_h = *Health_ rly* which is a query variable; C_m = *Command_ rly* which is an evidence variable; and C_d = *Closed_ rly* which is neither, but it cannot be pruned in this case, because it has descendent that is an evidence variable, namely the touch sensor variable *Sensor_ touchSens*.

8 Compilation to Arithmetic Circuits

We now very briefly summarize the compilation of Bayesian networks to arithmetic circuits (ACs). This compilation is the third stage in our approach to probabilistic model–based diagnosis. Prior to compilation, we modify the BN's CPTs to store pointers to AC nodes rather than numbers. For example, if 0.1 is stored in a particular slot of some CPT, then this number would be replaced with a pointer to a single AC node (sink) labeled with 0.1. Also prior to compilation, for each BN variable, we add a new table over just that variable representing the values of that variable. For example, variable X with values 0 and 1 would generate a table over X where the first slot contains a pointer to an AC node (sink) labeled with λ_0 and the second slot contains a pointer to an AC node (sink) labeled with λ_1.

After these two preprocessing steps, we run a slightly modified version of standard variable elimination (VE) [29, 30]. The only difference occurs when the standard version wishes to add or multiply two numbers. In each of these situations, the standard algorithm will identify two slots A and B in tables, add (multiply) the two numbers residing there, and store the result back into some slot C of some table. When the modified algorithm looks into A and B, it finds pointers to AC nodes α and β rather than numbers. Instead of performing the arithmetic operation, the modified algorithm creates a new AC node γ labeled with "+" or "*", makes α and β children of γ, and stores a pointer to γ into C. Upon completion, standard VE yields a single table containing a single slot containing a number. The modified algorithm will be the same, except that rather than a number, we will have a pointer to an AC node, which is the root of the compiled arithmetic circuit.

By exploiting local structure, this modified VE algorithm can yield an arithmetic circuit that is much smaller than exponential in treewidth. If one pays attention to how the CPTs of the Bayesian network representing EPSs are auto-generated as described in Section 7, it is easy to see that many of these CPTs

will be small and deterministic. Arithmetic circuit compilation has been shown to perform well on many such BNs, and as we shall see, the ADAPT BN is no exception [17, 44].

9 Experimental Results

Probabilistic inference is the fourth and final stage in our probabilistic model–based diagnosis approach, and the only one that needs to be performed on-line. We now discuss probabilistic inference experiments based on an ADAPT BN with 503 discrete nodes and 579 edges. In the ADAPT BN, the number of states per node ranges from 2 to 4 with an average of 2.23 and a median of 2. Experimental results as discussed here reflect that data was divided into two sets: real-world data from ADAPT and synthetic data that was automatically generated from the ADAPT BN. The purpose of the experiments with real-world data was to characterize the diagnostic quality of the ADAPT BN. The purpose of the experiments with synthetic data was to understand the performance of arithmetic circuit evaluation versus alternative BN inference algorithms, variable elimination and join tree propagation in particular. For arithmetic circuit evaluation, we used the ACE system to compile an ADAPT BN into an arithmetic circuit and to evaluate that arithmetic circuit (see `http://reasoning.cs.ucla.edu/ace/` regarding ACE). The timing measurements reported here were made on a PC with an Intel 4 1.83 GHz processor, 1 GB RAM, and Windows XP.

9.1 Experiments using Electrical Power System Data

9.1.1 Design

For experimentation using real-world data, EPS scenarios were generated using the ADAPT EPS at NASA Ames. This combined hardware and software facility is unique in its capability to produce real-world EPS data for testing diagnostic reasoners, while at the same time giving experimenters access to the gold standard (or true underlying state) for experiments. These scenarios, which are summarized in Table 7, cover component failures, sensor failures, and both component and sensor failures. In addition, each scenario contains one, two, or three faults. Finally, and in order to stress-test our probabilistic reasoner, we did not restrict inserted faults to discrete faults only. We also inserted continuous faults, specifically faults of the form "stuck at x", "noise StdDev = x", or "drift slope = x", with $x \in \mathbb{R}$. Since our probabilistic models do not contain continuous random variables, experiments with continuous faults cannot be diagnosed exactly, but they are still of great interest and included in many of the experiments reported on below.

In each scenario, ADAPT's initial state was as follows: Circuit breakers were commanded closed; the corresponding command variables in $\boldsymbol{E}_C$ were clamped to *cmdClose* in evidence $\boldsymbol{e}$. Relays were commanded open; the corresponding relay variables in $\boldsymbol{E}_C$ were clamped to *cmdOpen* in $\boldsymbol{e}$. In this initial state, all health nodes $\boldsymbol{H}$ are deemed healthy when computing $\text{MAP}(\boldsymbol{H})$, $\text{MAP}_{\text{MPE}}(\boldsymbol{H})$, or $\text{MAP}_{\text{MLV}}(\boldsymbol{H})$. Continuous data, in particular continuous sensor readings in $\boldsymbol{E}_C$, were discretized before being used for clamping the appropriate discrete random variables in the ADAPT model. To keep the experimental protocol consistent across scenarios, all inserted faults were persisted until the end of the experiments. The main diagnostic query $\text{MAP}_{\text{MPE}}(\boldsymbol{H})$, for which results are presented in Table 7, was also taken towards the end of a scenario.

9.1.2 Results

The results of the experiments with real-world data from ADAPT are summarized in Table 7. Each scenario is presented in one or more rows of the table, along with faults inserted and the diagnostic results computed for queries $\text{MAP}_{\text{MPE}}(\boldsymbol{H}, \boldsymbol{e})$. Since $\boldsymbol{H}$ contains 128 variables, reflecting the health status of 128 EPS components and sensors, we only show the variables found to be non-healthy in Table 7. Temporal progressions of sensor readings, for a varying subset of sensors, for eight of the sixteen scenarios are presented in Table 6.

9.1.3 Discussion

Our main observations regarding these experiments are as follows. We see in Table 7 that the different diagnostic queries correctly diagnose a majority of these component and sensor failure scenarios. In fact, there is an exact match in 10 of the 16 scenarios. Even in cases where there is not exact agreement, the diagnosis is either partly matching or at least reasonable as we will see in the following. We now discuss in more detail results of experiments, in particular experiments in which an exact match was not obtained.

In Experiment 305, a single failure to open for ESH175's feedback sensor was inserted, see Table 7. In other words, the relay sensor (or the touch sensor) says that relay ESH175 is open while it is in fact closed.

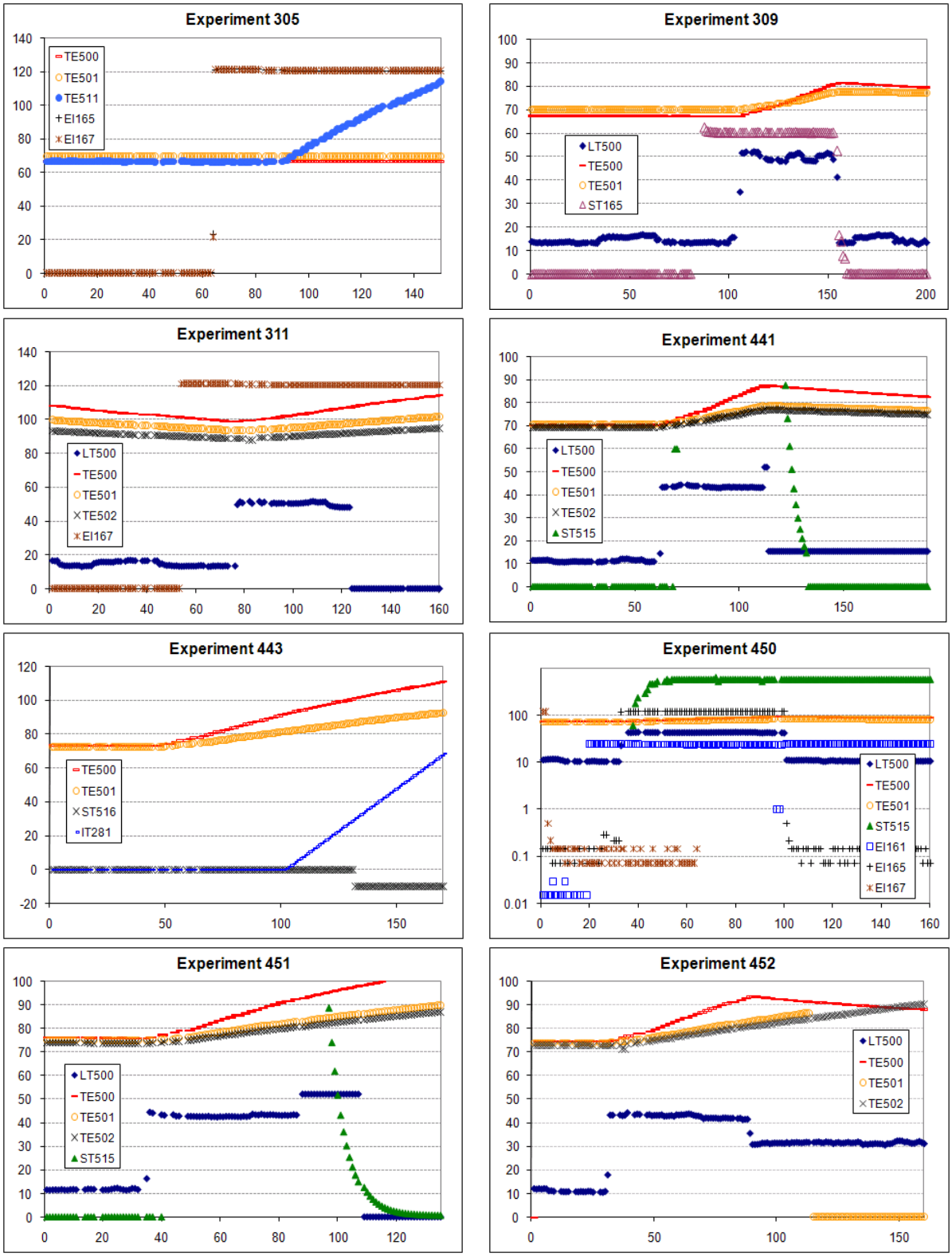

Table 6: Results for eight fault-injection experiments using ADAPT. Time, measured in sample number, is shown on the x-axis, while sampled sensor values for temperature, voltage, current, rotations per minute (RPMs), etc. are shown on the y-axis.

ID	Faults Inserted in ADAPT	Computed Diagnosis	Match
304	Relay EY260 failed open	*Health_relay_ey260_cl = stuckOpen*	Yes
305	Relay feedback sensor ESH175 failed open	*Health_relay_ey175_cl = stuckOpen*	Yes
306	Circuit breaker ISH262 tripped	*Health_breaker_ey262_op = stuckOpen*	Yes
308	Voltage sensor E261 failed low	*Health_e261 = stuckVoltageLo*	Yes
309	Battery BATT1 voltage low	*Health_battery1 = stuckDisabled*	Yes
310	Inverter INV1 failed off	*Health_inv1 = stuckOpen*	Yes
311	Light sensor LT500 failed low	*Health_lt500 = stuckLow*	Yes
441	Relay EY160 stuck open Big fan ST515 stuck at 0 RPM	*Health_relay_ey160_cl = stuckOpen*	Partly
442	Current sensor IT261 noise StdDev = 5 Relay feedback sensor ESH172 stuck at 0 Current sensor IT140 stuck at 100	*Health_it261 = stuckCurrentHi* *Health_esh172 = stuckOpen*	Partly
443	Current sensor IT281 drift slope = 2 Relay EY244 stuck closed Big fan ST516 stuck at -10 RPM	*Health_it281 = stuckCurrentHi* *Health_relay_ey244_cl = stuckClosed*	Partly
445	Voltage sensor E235 stuck at 0.3 Relay feedback sensor ESH344A stuck closed Inverter INV2 failed off	*Health_e235 = stuckVoltageLo* *Health_relay_ey344_cl = stuckClosed* *Health_inv2 = stuckOpen*	Partly
447	Voltage sensor E161 failed low Current sensor IT167 failed low	*Health_e161 = stuckVoltageLo* *Health_it167 = stuckCurrentLo*	Yes
449	Voltage sensor E140 failed low Voltage sensor E161 failed low	*Health_e140 = stuckVoltageLo* *Health_e161 = stuckVoltageLo*	Yes
450	Inverter INV1 failed off Big fan ST515 stuck at 600 RPM	*Health_inv1 = stuckOpen* *Health_fan1_speed_st515 = stuckMid*	Partly
451	Relay EY171 failed open Light sensor LT500 failed low	*Health_relay_ey171_cl = stuckOpen* *Health_lt500 = stuckLow*	Yes
452	Light bulb TE500 failed off Temperature sensor TE501 failed low	*Health_load170_bulb1 = stuckDisabled*	Partly

Table 7: Inserted faults versus diagnostic results — computed using the most probable diagnosis $\mathrm{MAP}_{\mathrm{MPE}}(\boldsymbol{H}, \boldsymbol{e})$ — for different fault scenarios (with IDs 304, 305, ...) for the electrical power systemtestbed ADAPT.

Since this and the other relays upstream of LGT4 are closed, LGT4 is powered as indicated by temperature sensor TE511's increasing temperature — starting at time 90 — as shown in Table 6. ACE computed a correct diagnosis ESH175 relay sensor failed or stuck open, see Table 7.

In Experiment 309, a single battery failure was inserted in BATT1. Specifically, the battery BATT1 failed with low voltage around time 150. The effect of this failure is a general drop in light, temperature, and RPMs as reflected in Table 6. This fault is correctly diagnosed by ACE (see Table 7).

In Experiment 311, LT500's light sensor was failed low around time 120. This is reflected in Table 6 as follows: The AC voltage from the inverter goes high around time 55, and the light sensor and the temperature sensors indicate that the light bulbs are on starting at time 80. A little after time 120, LT500 goes to zero, however. Since the temperatures appear to continue to rise (see TE500, TE501, and TE502 in Table 6), a light sensor fault as computed by ACE is very reasonable and in fact the correct diagnosis.

In Experiment 441 in Table 7, both EY160 and ST515 were failed. However, one of these faults (namely relay EY160 stuck open) is sufficient to explain all observations. The reason for this is that EY160 controls power to all loads on load bank 1, including the fan sensed by ST515. Since EY160 is upstream of and controls the power to ST515, see Figure 3, the stuck at 0 fault of ST515 is consistent with the single-fault EY160 stuck open diagnosis computed by ACE, and in fact has a greater probability than the double faults actually inserted. In other words, the ST515 failure is masked by the EY160 failure.

Experiments 442 and 443 have continuous faults inserted (change in noise StdDev and drift respectively) that are currently beyond the scope of our discrete probabilistic model. In Experiment 442, there are no sensors on the affected load, making it difficult to detect whether (i) the relay has failed open, thus turning off the load, or (ii) the relay feedback sensor has failed open. (The BN is currently not modelling the exact current draw in the system). So, if the relay failed open and turned off the attached load there would be

a drop in current being drawn from the battery because there are fewer loads. But we are not discretizing the current nodes in this way, sometimes making it difficult to distinguish between relay failure and relay position sensor failure.

In Experiment 443, IT281 starts drifting at time 100 and ST516 gets stuck at -10 RPM at time 130. Both of these failure as easy to see in Table 6. Here, 2 of the 3 faulty components were correctly isolated in spite of continuous faults being inserted. We now consider the one fault not caught: This fault was inserted in ST516, a fan on Load bank 2, which was not commanded on during this experiment. In other words, the fact that ST516 was neither on nor commanded to be on made the abnormally low sensor reading of -10 RPM harder to detect. Another issue was the discretization in the BN, where the faulty sensor reading of -10 is binned with the correct sensor reading of 0. Thus, even though the diagnosis misses ST516's failure, a more fine-grained discretization could have caught this.

In Experiment 445, 2 of the 3 faulty components were correctly isolated, and the only difficulty was due to the continuous fault inserted, namely E235 stuck at 0.3, discretized in the AC to stuck low.

In Experiment 450, two faults were inserted: inverter INV1 failed and fan ST515 got stuck at 500RPM. When the inverter failed, all downstream power was disrupted. So E165, E167, ST165, LT500, IT167 go to low values. The inverter failure around time 100 is perhaps easiest to see in Table 6 if we compare the voltage downstream of the inverter (EI165 and EI167 both show minimal voltage after around time 100) with the voltage upstream of the inverter (EI161 shows nominal voltage). On the load side, we note that TE500 and TE501 are increasing till around time 100, at which time they start decreasing. This behavior is also consistent with INV1 failure at time 100; the diagnosis *Health_inv1 = stuckOpen* is correct. The failure of ST515, which is powered through INV1, is clearly visible in Table 6. ST515, which should also have gone to a low value (because the fan does not have power anymore and therefore is not spinning), was stuck reading that its nominal value was around 600, which lead to the partly correct diagnosis *stuckMid*. The diagnosis is here essentially correct, however due to the discretization the diagnosis is approximate. A more fine-grained discretization could have improved the approximation.

In Experiment 451, two faults EY171 failed open and LT500 failed low were inserted. The LT500 fault manifested itself in the drop of the measured lighting level for LT500 around time 110. However, the temperatures (as measured by TE500, TE501, and TE502) keep rising, suggesting power is still flowing. The EY171 fault manifested itself in ST515's dropping RPM around time 100; clearly, EY171 failed open is an explanation of this, since EY171 controls ST515. ACE computed the correct diagnosis of LT500 sensor failure and EY171 failed open.

In Experiment 452, two faults were inserted, for Bulb 0 (with sensor TE500) and TE501 (for Bulb 1). Bulb 0 was failed off, while TE501 was failed low. Towards the end of this experiment, light sensor LT500 falls from ≈ 43 to ≈ 32. This lower value of ≈ 32 strongly suggests that only two bulbs are on, in other words that one bulb, out of the three bulbs present, had failed. Temperature sensor TE500 started falling, indicating that the bulb associated with that sensor was off. Then TE501 went to 0 while the light sensor reading remained the same, indicating that temperature sensor TE501 was likely also faulty. At the time of the diagnosis, we have the following evidence: TE500 reads *high* (however, its derivative is negative indicating that the bulb is off – but that is not in the discrete model); TE501 reads *low*; TE502 reads *high*; and LT500's sensor reading indicates that two bulbs are lit. Thus, based on the evidence provided to ACE, it finds a single fault of *stuckDisabled* for Bulb 1. This diagnosis of *Health_load170_bulb1 = stuckDisabled* is a direct result of the TE501 and LT500 readings. However, what was inserted was two faults, for Bulb 0 and TE501. This highlights two issues. First, the discretization does not perfectly capture the signature of a bulb being *off*. Specifically, the bulb is still warm from having previously been on, leading the TE500 value to be above the threshold defined for *on*. A second issue is that temporal aspects are not captured by taking one time slice near end of run; in this case there are temporal clues that point toward the correct diagnosis.

We note that there are several different but related phenomena underlying the mismatches in Table 7. Due to our knowledge of the faults inserted into ADAPT, we are in a position to discuss these different phenomena in more detail. First, and reflecting the challenging nature of the fault scenarios that can be created using ADAPT, continuous faults (as inserted in experiments 441, 442, 443, 445, 448, and 450) are simply beyond the scope of our currently discrete probabilistic model. A second phenomenon is the following. A probabilistic model allows one to distinguish between what is possible versus impossible, and among what is possible, compute the probabilities for different possible explanations. However, there is no guarantee that the inserted faults are part of a unique MPE for the given evidence for $\boldsymbol{E}$. For example, three faults may have been inserted into ADAPT, but there may be an explanation with one or two faults that has the same or higher joint probability, given the evidence for those faults. Experiment 441 is a good example of

Inference	MPE		Marginals	
Time (ms)	VE	ACE	JTP	ACE
Minimum	19.30	0.2235	9.792	0.5721
Maximum	40.21	2.5411	65.34	5.9228
Median	19.81	0.2260	10.52	0.6006
Mean	20.13	0.2625	11.01	0.7854
St. Dev.	1.554	0.2028	4.101	0.6970

Table 8: Results for different inference algorithms (VE, ACE, and CTP) when computing MPEs and marginals using synthetic data generated from the ADAPT BN.

this effect. Third, there are faults that could have been detected had more fine-grained discretizations of random variables been used. Experiment 452 provides an example of this, since the drop in the reading of the temperature sensor TE501 was quite dramatic and indicative of a sensor failure rather than only a failure in the light bulb TE500. A fourth phenomenon is that there might be too few, improperly placed, or inadequate sensors to distinguish between different faults. Many of the mismatches in these experiments could have been detected had more appropriate sensing been used; a detailed discussion of sensor placement is beyond the scope of this work, however.

In summary, we have observed strong performance for our probabilistic model in these controlled experiments with ADAPT. We also note that a richer way of presenting diagnostic results would be helpful but non-trivial to provide. Specifically, it would be useful to have access to all non-zero explanations and their probabilities, not just the most probable explanation but explanations with lower probabilities. These experimental results also motivate several future research directions as discussed in Section 10.

9.2 Experiments using Simulated Data

In order to understand how arithmetic circuit evaluation performs in comparison to other BN inference algorithms in the ADAPT setting, a large number of scenarios were automatically generated and used in experiments as discussed in the following.

9.2.1 Design

In order to better understand the performance of arithmetic circuit evaluation (ACE), we performed comparative experiments with variable elimination (VE) and join tree propagation (JTP). Simulated data was created by a program that (i) generated a set of failure scenarios according to the probabilities of the ADAPT BN's health nodes $\boldsymbol{H}$, and (ii) generated evidence by doing stochastic simulation for each failure scenario. These evidence sets were then used as evidence in the three different inference systems, and inference was performed as presented below.

9.2.2 Results

Results from the experiments are summarized in Table 8. Both MPEs and marginals were computed for 200 simulated evidence sets generated from the ADAPT BN.

9.2.3 Discussion

The main points, which are in line with previous results on a smaller version of the ADAPT BN [45], are as follows. On average, ACE is over 76 times faster than VE when computing MPEs (see Table 8). In addition, ACE can compute all marginals, supporting the probabilistic queries BEL(H, $\boldsymbol{e}$) (where $H \in \boldsymbol{H}$) and $\text{MAP}_{\text{MLV}}(\boldsymbol{H}, \boldsymbol{e})$, using just slightly more than twice the time used for computing MPEs, or $\text{MAP}_{\text{MPE}}(\boldsymbol{H}, \boldsymbol{e})$. In other words, ACE computes probabilities over 500 random variables more than 33 times faster than VE computes probabilities for a single random variable. The third inference system, JTP, can compute all marginals in a manner similar to ACE. This overcomes VE's limitation of computing probabilities for only one random variable at a time. Compared to ACE, however, JTP is over 14 times slower and also has a standard deviation that is more than 5 times greater.

The auto-generation approach produces, for ADAPT, a BN that all three systems perform well on. This illustrates that the ADAPT BN was carefully generated, using our novel modelling approach and auto-generation algorithm, in a manner that supports efficient inference using three quite different exact inference

algorithms. Our two next points consider the performance of ACE versus VE and JTP. First, the ACE system outperforms VE for MPE computation and JTP for computation of marginals; in fact ACE is one or two orders of magnitude more efficient than these two other algorithms. Second, the standard deviation is substantially smaller for ACE than for VE and JTP. The fast and predictable inference times of ACE are both very important factors for electrical power system health management in the real-time setting of aerospace.

10 Conclusion and Future Work

In this work, we have discussed an electrical power system application of the probabilistic approach to model-based diagnosis. Specifically, we have discussed the use of Bayesian networks and arithmetic circuits to perform diagnosis and health management in electrical power systems in aircraft and spacecraft. We have emphasized two important issues that arise in engineering diagnostic applications in this area, namely the challenges of modelling and real-time reasoning. The modelling challenge concerns how to model a real-world EPS by means of Bayesian networks. To address this challenge, we developed a systematic way of representing electrical power systems as Bayesian networks, supported by an easy-to-use specification language and an auto-generation algorithm. The second challenge, that of real-time reasoning, is associated with the embedding of algorithms that solve computationally hard problems, including diagnostic reasoning, into hard real-time systems [14, 15]. To address this challenge, we compile Bayesian networks into arithmetic circuits, an approach that supports real-time diagnosis in two ways. First, the use of arithmetic circuits results in more predictable diagnostic inference times. Second, it results in much faster inference.

While compilation of Bayesian networks to arithmetic circuits is well-established [37, 39, 38, 40, 17], this work further extends the reach of the technology by introducing a high-level EPS specification languages from which Bayesian networks are auto-generated, and showing that the combined approach gives strong results on a real-world EPS.

Future directions of work include the following. First, improved modeling of and reasoning with continuous behavior, using soft evidence, highly discretized, and/or continuous random variables, along with representation using arithmetic circuits for purposes of compilation, would be of great interest. A second area of interest is improved modeling of dynamic, transient, and cascading faults along with their integration into the compilation approach. Third, it would be very useful to extend the high-level specification language to handle (i) novel components and states; (ii) continuous behavior; and (iii) dynamics such as transient and cascading faults. Finally, it would be interesting to further investigate sensing issues, including the questions of optimal sensor placement as well as the number and types of sensors needed to distinguish between different faults.

Acknowledgments

This material is based upon work supported by NASA under awards NCC2-1426 and NNA07BB97C. We would also like to thank Ann Patterson-Hine and Dougal Maclise (NASA Ames Research Center) for their central roles in the development of the ADAPT testbed, and David Garcia and David Nishikawa (NASA Ames Research Center) for generating the ADAPT data for many of the experiments discussed here. Other members of the ADAPT team are also acknowledged for their contributions.

References

[1] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. J. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, A. Sweet, S. Yentus, I. Roychoudhury, M. Daigle, G. Biswas, and X. Koutsoukos, "Advanced diagnostics and prognostics testbed," in *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, (Nashville, TN), pp. 178–185, 2007.

[2] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.

[3] U. Lerner, R. Parr, D. Koller, and G. Biswas, "Bayesian fault detection and diagnosis in dynamic systems," in *Proceedings of the Seventeenth national Conference on Artificial Intelligence (AAAI-00)*, pp. 531–537, 2000.

[4] C.-F. Chien, S.-L. Chen, and Y.-S. Lin, "Using Bayesian network for fault location on distribution feeder," *IEEE Transactions on Power Delivery*, vol. 17, pp. 785–793, 2002.

[5] Z. Yongli, H. Limin, and L. Jinling, "Bayesian network-based approach for power system fault diagnosis," *IEEE Transactions on Power Delivery*, vol. 21, pp. 634–639, 2006.

[6] S. Andreassen, M. Woldbye, B. Falck, and S. Andersen, "MUNIN – A causal probabilistic network for interpretation of electromyographic findings," in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, (Milan, Italy), pp. 366–372, August 1987.

[7] M. Shwe, B. Middleton, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper, "Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base: I. The probabilistic model and inference algorithms," *Methods of Information in Medicine*, vol. 30, no. 4, pp. 241–255, 1991.

[8] N. Chater and C. D. Manning, "Probabilistic models of language processing and acquisition," *TRENDS in Cognitive Sciences*, vol. 10, no. 7, pp. 335–344, 2006.

[9] H. Langseth and L. Portinale, "Bayesian networks in reliability," *Reliability Engineering and System Safety*, vol. 92, no. 1, pp. 92–108, 2007.

[10] P. Jones, C. Hayes, D. Wilkins, R. Bargar, J. Sniezek, P. Asaro, O. J. Mengshoel, D. Kessler, M. Lucenti, I. Choi, N. Tu, and J. Schlabach, "CoRAVEN: Modeling and design of a multimedia intelligent infrastructure for collaborative intelligence analysis," in *Proceedings of the International Conference on Systems, Man, and Cybernetics*, (San Diego, CA), pp. 914–919, October 1998.

[11] C. C. Ruokangas and O. J. Mengshoel, "Information filtering using Bayesian networks: Effective user interfaces for aviation weather data," in *Proceedings of the 2003 International Conference on Intelligent User Interfaces*, (Miami, FL), pp. 280–283, 2003.

[12] R. G. Gallager, "Low density parity check codes," *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, Jan 1962.

[13] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge, UK: Cambridge University Press, 2002.

[14] D. Musliner, J. Hendler, A. K. Agrawala, E. Durfee, J. K. Strosnider, and C. J. Paul, "The challenges of real-time AI," *IEEE Computer*, vol. 28, pp. 58–66, January 1995.

[15] O. J. Mengshoel, "Designing resource-bounded reasoners using Bayesian networks: System health monitoring and diagnosis," in *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, (Nashville, TN), pp. 330–337, 2007.

[16] A. Darwiche, "A differential approach to inference in Bayesian networks," *Journal of the ACM*, vol. 50, no. 3, pp. 280–305, 2003.

[17] M. Chavira and A. Darwiche, "Compiling Bayesian networks using variable elimination," in *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, (Hyderabad, India), pp. 2443–2449, 2007.

[18] R. M. Button and A. Chicatelli, "Electrical power system health management," in *Proceedings of the 1st International Forum on Integrated System Health Engineering and Management in Aerospace*, (Napa, CA), 2005.

[19] F. G. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *Artificial Intelligence*, vol. 42, pp. 393–405, 1990.

[20] E. Shimony, "Finding MAPs for belief networks is NP-hard," *Artificial Intelligence*, vol. 68, pp. 399–410, 1994.

[21] J. D. Park and A. Darwiche, "Complexity results and approximation strategies for MAP explanations," *Journal of Artificial Intelligence Research (JAIR)*, vol. 21, pp. 101–133, 2004.

[22] S. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems (with discussion)," *Journal of the Royal Statistical Society series B*, vol. 50, no. 2, pp. 157–224, 1988.

[23] S. K. Andersen, K. G. Olesen, F. V. Jensen, and F. Jensen, "HUGIN—a shell for building Bayesian belief universes for expert systems," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, vol. 2, (Detroit, MI), pp. 1080–1085, Aug. 1989.

[24] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen, "Bayesian updating in causal probabilistic networks by local computations," *SIAM Journal on Computing*, vol. 4, pp. 269–282, 1990.

[25] P. P. Shenoy, "A valuation-based language for expert systems," *International Journal of Approximate Reasoning*, vol. 5, no. 3, pp. 383–411, 1989.

[26] J. Pearl, "A constraint - propagation approach to probabilistic reasoning," in *Uncertainty in Artificial Intelligence* (L. N. Kanal and J. F. Lemmer, eds.), pp. 357–369, Amsterdam, Netherlands: Elsevier, 1986.

[27] A. Darwiche, "Recursive conditioning," *Artificial Intelligence*, vol. 126, no. 1-2, pp. 5–41, 2001.

[28] Z. Li and B. D'Ambrosio, "Efficient inference in Bayes nets as a combinatorial optimization problem," *International Journal of Approximate Reasoning*, vol. 11, no. 1, pp. 55–81, 1994.

[29] N. L. Zhang and D. Poole, "Exploiting causal independence in Bayesian network inference," *Journal of Artificial Intelligence Research*, vol. 5, pp. 301–328, 1996.

[30] R. Dechter, "Bucket elimination: A unifying framework for reasoning," *Artificial Intelligence*, vol. 113, no. 1-2, pp. 41–85, 1999.

[31] K. Kask and R. Dechter, "Stochastic local search for Bayesian networks," in *Proceedings Seventh International Workshop on Artificial Intelligence and Statistics*, (Fort Lauderdale, FL), Morgan Kaufmann, Jan 1999.

[32] O. J. Mengshoel, *Efficient Bayesian Network Inference: Genetic Algorithms, Stochastic Local Search, and Abstraction*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, Apr. 1999.

[33] O. J. Mengshoel, D. Roth, and D. C. Wilkins, "Stochastic greedy search: Computing the most probable explanation in Bayesian networks," Tech. Rep. UIUCDCS-R-2000-2150, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, Feb. 2000.

[34] F. Hutter, H. H. Hoos, and T. Stützle, "Efficient stochastic local search for MPE solving," in *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, (Edinburgh, Scotland), pp. 169–174, 2005.

[35] O. J. Mengshoel, "Understanding the role of noise in stochastic local search: Analysis and experiments," *Artificial Intelligence*, vol. 172, no. 8-9, pp. 955–990, 2008.

[36] J. D. Park and A. Darwiche, "Approximating MAP using local search," in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-01)*, (Seattle, WA), pp. 403–410, 2001.

[37] A. Darwiche, "A differential approach to inference in Bayesian networks," in *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI)*, pp. 123–132, 2000.

[38] J. D. Park and A. Darwiche, "A differential semantics for jointree algorithms," *Artificial Intelligence*, vol. 156, no. 2, pp. 197–216, 2004.

[39] A. Darwiche, "A logical approach to factoring belief networks," in *Proceedings of the Eight International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pp. 409–420, 2002.

[40] M. Chavira and A. Darwiche, "Compiling Bayesian networks with local structure," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1306–1312, 2005.

[41] D. R. Bromaghim, J. R. Leduc, R. M. Salasovich, G. G. Spanjers, J. M. Fife, M. J. Dulligan, J. H. Schilling, D. C. White, and L. K. Johnson, "Review of the electric propulsion space experiment (esex) program," *Journal of Propulsion and Power*, vol. 18, no. 4, pp. 723–730, 2002.

[42] O. J. Mengshoel, A. Darwiche, and S. Uckun, "Sensor validation using Bayesian networks," in *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS-08)*, 2008.

[43] R. D. Shachter, "Evaluating influence diagrams," *Operations Research*, vol. 34, no. 6, pp. 871–882, 1986.

[44] M. Chavira, *Beyond Treewidth in Probabilistic Inference*. PhD thesis, University of California, Los Angeles, 2007.

[45] O. J. Mengshoel, A. Darwiche, K. Cascio, M. Chavira, S. Poll, and S. Uckun, "Diagnosing faults in electrical power systems of spacecraft and aircraft," in *Proceedings of the Twentieth Innovative Applications of Artificial Intelligence Conference (IAAI-08)*, (Chicago, IL), pp. 1699–1705, 2008.